Haskell-style Overloading is NP-hard

Dennis M. Volpano †
Department of Computer Science
Naval Postgraduate School
Monterey, California, 93943
volpano@cs.nps.navy.mil

Abstract

Extensions of the ML type system, based on constrained type schemes, have been proposed for languages with overloading. Type inference in these systems requires solving the following satisfiability problem. Given a set of type assumptions C over finite types and a type basis A, is there is a substitution S that satisfies C in that $A \vdash CS$ is derivable? Under arbitrary overloading, the problem is undecidable. Haskell limits overloading to a form similar to that proposed by Kaes called parametric overloading. We formally characterize parametric overloading in terms of a regular tree language and prove that although decidable, satisfiability is NP-hard when overloading is parametric.

1 Introduction

A practical limitation of the ML type system is that it prohibits *global overloading* in a programming language by restricting to at most one the number of assumptions per identifier in a type context, a limitation noted by Milner himself [Mil78]. Suppose we wish to assert that a free identifier, say $+$, has precisely finite types $int \to int \to int$ and $real \to real \to real$. Any context in which $+$ has one of the two desired finite types precludes a derivation that it has the other. On the other hand, any context that assigns type scheme $\forall\alpha.\alpha \to \alpha \to \alpha$ to $+$ is one from which too many types can be derived for $+$. There is no type context in system ML from which we can derive all and only the desired finite types for $+$. Even system ML with subtypes is inadequate. From type context

$$A = \{int \subseteq real,\ \ + : real \to real \to real\}$$

one could derive $A \vdash + : int \to int \to real$ but not $A \vdash + : int \to int \to int$.

Several type disciplines have emerged for programming languages with overloading. Among them are those based on *intersection types* [CoD78, Sal78, CDV80] and those based on *constrained type schemes*, the latter being inspired by the design of Haskell [WaB89, CDO91, Smi91, Kae92, CHO92, Jon92]. The type system of Forsythe, an explicitly-typed descendant of Algol, is based on an intersection type discipline, namely $\lambda_\wedge$ [Rey88]. Though useful, $\lambda_\wedge$ remains limited in that it has no type schemes and all intersections are finite [Lie90, Pie91].

A more flexible type discipline for languages with overloading is an extension of the ML type system with *constrained type schemes* [Kae92, Smi93]. Using the notation of [Smi93], a constrained type scheme has the general form

$$\forall\alpha_1, \ldots, \alpha_n \textbf{ with } x_1 : \tau_1, \ldots, x_m : \tau_m \,.\, \tau$$

where τ is a finite type. Finite types are defined in the usual way. Every type variable α is a finite type, and if $\tau_1, \ldots, \tau_n$ are finite types then so are $\tau_1 \to \tau_2$ and $\chi(\tau_1, \ldots, \tau_n)$ where χ is a type constructor of arity n. The $x_1 : \tau_1, \ldots, x_m : \tau_m$ are *constraints* on overloaded free identifiers $x_1, \ldots, x_m$. Quantifier $\forall$ is omitted if there are no quantified variables and the **with** clause is omitted if there are no constraints, in which case we have an ordinary ML type scheme. Unlike the ML type system, a free identifier may be overloaded, that is, have multiple assumptions in an initial type context, so we refer to this extension as system ML_o.

The fact that a free identifier is permitted to have more than one assumption in a type context immediately raises the issue of semantic ambiguity in terms. Care must be taken to ensure that terms with overloaded identifiers have unambiguous meaning. Consider, for instance, type context

$$\left\{ \begin{array}{l} + : real \to real \to real, \\ + : \forall\alpha.set(\alpha) \to set(\alpha) \to set(\alpha) \end{array} \right\} \qquad (1)$$

†Appeared in Proc. 1994 Int'l Conference on Computer Languages, Toulouse, France, pp.88–95, 16–19 May 1994.

Report Documentation Page

Form Approved
OMB No. 0704-0188

1. REPORT DATE	2. REPORT TYPE	3. DATES COVERED
01 MAY 1994	**N/A**	**-**

4. TITLE AND SUBTITLE	5a. CONTRACT NUMBER
Haskell-style Overloading is NP-hard	5b. GRANT NUMBER
	5c. PROGRAM ELEMENT NUMBER
6. AUTHOR(S)	5d. PROJECT NUMBER
	5e. TASK NUMBER
	5f. WORK UNIT NUMBER
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) **Department of Computer Science Naval Postgraduate School Monterey, CA 93943**	8. PERFORMING ORGANIZATION REPORT NUMBER
9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)	10. SPONSOR/MONITOR'S ACRONYM(S)
	11. SPONSOR/MONITOR'S REPORT NUMBER(S)

12. DISTRIBUTION/AVAILABILITY STATEMENT
Approved for public release, distribution unlimited

13. SUPPLEMENTARY NOTES

14. ABSTRACT

15. SUBJECT TERMS

16. SECURITY CLASSIFICATION OF:			17. LIMITATION OF ABSTRACT	18. NUMBER OF PAGES	19a. NAME OF RESPONSIBLE PERSON
a. REPORT	b. ABSTRACT	c. THIS PAGE			
unclassified	**unclassified**	**unclassified**	**UU**	**7**	

where + denotes *real* addition and set union. If one can derive from this context that term $\lambda x.\, x + x$ has type $real \to real$ and $\forall\beta.set(\beta) \to set(\beta)$ then the term can be interpreted in one of two different ways. Its meaning then must be determined by a process called *overloading resolution* whose outcome depends on the type of x. Thus we say that the overloading of + above is *incoherent*. Surprisingly, incoherent overloading is pervasive among languages despite its potential for semantic ambiguity. For example, "/" is often overloaded, as in Ada, to stand for integer and floating-point division.

Coherent overloading, on the other hand, gives rise to *discrete polymorphism* where the meaning of a term does not depend on overloading resolution. In this setting, a semantics for an operator, say f, is postulated as a set of sentences, or axioms, Δ say in first-order logic. A model of Δ is any interpretation that satisfies it. So if for a type basis A, one is able to derive $A \vdash f : \sigma_1$ and $A \vdash f : \sigma_2$, then the overloading of f within A is coherent if σ_1 and σ_2 are each models of Δ. If so, then we may regard f as belonging to the intersection of σ_1 and σ_2. Coherent overloading then allows the meaning of every term to be uniquely determined by simply appealing to the axioms for the operators in question which, after all, is where semantics should be prescribed.

For instance, suppose $\Delta = \{\forall x.\, x+x = x\}$. Both an interpretation of + as set union and logical disjunction satisfy Δ, so sets and truth values are models of Δ. But Δ is false under a real number interpretation. So we would regard the sentence in Δ as an axiom of set theory and boolean algebra, but not the first-order theory of reals with addition. The overloading of + then in set (1) is incoherent if we adopt the sentence as an axiom of our intended meaning of +. However if the first assumption for + in (1) is replaced by $+ : bool \to bool \to bool$ then the overloading is coherent. So although we may be able to derive from (1) that $\lambda x.\, x+x$ has type $bool \to bool$ and $\forall\beta.set(\beta) \to set(\beta)$, we know the function belongs to the intersection of the two types and its meaning, given uniquely by Δ, is a function that behaves as the identity function.

1.1 Satisfiability

Two new type assignment rules, (∀-intro) and (∀-elim) given in Figure 1, accompany constrained types. For a constraint set C, the notation $A \vdash C$ means that for each constraint $x : \tau$ in C, $A \vdash x : \tau$ is derivable. The notation $[\bar\alpha := \bar\tau]$ denotes a substitution, the application of which is written in postfix form. Observe that when C is empty the two rules reduce

$$\text{(∀-intro)}\quad \frac{A \cup C \vdash M : \tau',\quad A \vdash C[\bar\alpha := \bar\tau],\ \ \bar\alpha \text{ not free in } A}{A \vdash M : \forall\bar\alpha\ \mathbf{with}\ C\,.\,\tau'}$$

$$\text{(∀-elim)}\quad \frac{A \vdash M : \forall\bar\alpha\ \mathbf{with}\ C\,.\,\tau',\quad A \vdash C[\bar\alpha := \bar\tau]}{A \vdash M : \tau'[\bar\alpha := \bar\tau]}$$

Figure 1: Generalization and specialization in $ML_\circ$

respectively to type generalization and instantiation in system ML [Mil78, DaM82].

The antecedent of (∀-intro) requires C be satisfiable with respect to A. That is, for some finite types $\bar\tau$, $A \vdash C[\bar\alpha := \bar\tau]$ must be derivable. Operators that are constrained in C and interact share a type variable which in essence hypothesizes a model common to their semantics. Satisfiability of C then ensures the existence of such a model assuming overloading is coherent. If a model exists (there may be more than one), then the meaning of M is uniquely determined by the axioms of the operators, otherwise M has no meaning and consequently should be and is untypable. For example, suppose

$$\left\{ \begin{array}{l} +\ : bool \to bool \to bool, \\ +\ : \forall\alpha.set(\alpha) \to set(\alpha) \to set(\alpha) \end{array} \right\} \qquad (2)$$

is a coherent overloading with respect to semantics $\Delta_1 = \{\forall x.\, x + x = x\}$ and suppose

$$\left\{ \begin{array}{l} \leq : int \to int \to bool, \\ \leq : \forall\alpha.set(\alpha) \to set(\alpha) \to bool \end{array} \right\} \qquad (3)$$

is a coherent overloading relative to an axiomatization, say Δ_2, of a partial order. We can derive from (2)∪(3) that $\lambda x.\,(x + x) \leq x$ has type $\forall\alpha.set(\alpha) \to bool$ since Δ_1 and Δ_2 have sets as a common model. So the meaning of the term is given by Δ_1 and Δ_2 and is a constant function mapping sets into **true**. If there were no common model then the axioms could not be applied and the term would be meaningless.

So rules (∀-intro) and (∀-elim) give rise to the following satisfiability problem.

Definition 1.1 *The problem of constraint-set satisfiability CS-SAT is deciding for a given set of type assumptions C, involving only finite types (constraints), and an assumption set A, whether there is a substitution S such that $A \vdash CS$ is derivable.*

Without any restrictions on the kind of overloading in A, *CS-SAT* is undecidable [Smi91]. Constrained type schemes permit recursive overloadings where an

$$
\begin{array}{l}
+ : \mathit{real} \rightarrow \mathit{real} \rightarrow \mathit{real} \\
+ : \forall\alpha \ \textbf{with} + : \alpha \rightarrow \alpha \rightarrow \alpha \, . \\
\qquad \mathit{matrix}(\alpha) \rightarrow \mathit{matrix}(\alpha) \rightarrow \mathit{matrix}(\alpha) \\
* : \mathit{int} \rightarrow \mathit{int} \rightarrow \mathit{int} \\
* : \mathit{real} \rightarrow \mathit{real} \rightarrow \mathit{real} \\
* : \forall\alpha \ \textbf{with} + : \alpha \rightarrow \alpha \rightarrow \alpha, * : \alpha \rightarrow \alpha \rightarrow \alpha \, . \\
\qquad \mathit{matrix}(\alpha) \rightarrow \mathit{matrix}(\alpha) \rightarrow \mathit{matrix}(\alpha)
\end{array}
$$

Figure 2: A recursive overloading

assumption for an overloaded identifier has a constraint whose satisfiability may depend on the assumption itself. This permits type assumptions to be very expressive. For example, + and $*$ are overloaded recursively in Figure 2 due to constraints on + and $*$. Eliminating recursion altogether makes *CS-SAT* decidable but this is unacceptable because it arises naturally in practice as Figure 2 shows. Smith gives a restriction called *overloading by constructors* that allows *CS-SAT* to be solved in polynomial time [Smi91]. But it prohibits the kind of recursion given in Figure 2. The functional language Haskell adopts another restriction similar to that proposed by Kaes called *parametric overloading* [Kae88].

2 Parametric Overloading

Assumption sets that arise in practice often follow a very simple pattern of overloading called parametric overloading [Kae88]. This form of overloading allows natural recursive overloadings and makes *CS-SAT* decidable. To define it, we introduce the notion of the *least common generalization* (*LCG*) of a set of finite types which captures common structure among type assumptions for overloaded identifiers [Rey70, McC84].

Definition 2.1 *A finite type τ is a common generalization of finite types $\tau_1, \ldots, \tau_n$ if there are n substitutions $S_1, \ldots, S_n$ such that $\tau S_i = \tau_i$ for all i; τ is the least common generalization of these types if in addition there is a substitution S such that $\tau' S = \tau$ for any other generalization τ'.*

It is useful to extend this definition to identifiers. If identifier x is overloaded with constrained type schemes $\forall\bar{\gamma}_1$ **with** $C_1 \, . \, \tau_1, \ldots, \forall\bar{\gamma}_n$ **with** $C_n \, . \, \tau_n$, such that $\tau_1, \ldots, \tau_n$ has τ as *LCG* with free variables $\bar{\alpha}$, then $\forall\bar{\alpha} . \, \tau$ is the *LCG* of x.

For example, if + is overloaded with assumptions $+ : \mathit{int} \rightarrow \mathit{real} \rightarrow \mathit{real}$ and $+ : \mathit{real} \rightarrow \mathit{complex} \rightarrow \mathit{complex}$ then its *LCG* is $\forall\alpha, \beta . \, \alpha \rightarrow \beta \rightarrow \beta$.

$$
\begin{array}{l}
f, \, g : \chi_1 \\
f : \forall\alpha \ \textbf{with} \ g : \alpha \, . \, \chi_2(\alpha) \\
g : \forall\alpha \ \textbf{with} \ f : \alpha \, . \, \chi_3(\alpha)
\end{array}
$$

Figure 3: A mutually-recursive overloading

Definition 2.2 *Parametric assumption sets are defined inductively.*

The empty set is parametric.

If A is parametric with no assumption for x and σ is a constrained type scheme $\forall\alpha$ **with** $C \, . \, \tau$ such that for each $z : \rho \in C$, z is overloaded in A and ρ is a generic instance of its *LCG* then $A \cup \{x : \sigma\}$ is parametric.

If A is parametric with no assumption for x and B is the set

$$
\left\{ \begin{array}{c}
x : \forall\bar{\gamma}_1 \ \textbf{with} \ C_1 \, . \, \tau[\alpha := \chi_1(\bar{\gamma}_1)] \\
\vdots \\
x : \forall\bar{\gamma}_n \ \textbf{with} \ C_n \, . \, \tau[\alpha := \chi_n(\bar{\gamma}_n)]
\end{array} \right\}
$$

such that

- x has *LCG* $\forall\alpha . \, \tau$,
- $\chi_i \neq \chi_j$ for $i \neq j$, and
- $z : \rho \in C_i$ implies that z has *LCG* $\forall\pi . \, \rho$, for some $\pi \in \bar{\gamma}_i$, and either z is overloaded in A or $z = x$,

then $A \cup B$ is parametric.

Examples of parametric assumption sets are given below and in Figure 2.

$$
\left\{ \begin{array}{l}
= : \mathit{int} \rightarrow \mathit{int} \rightarrow \mathit{bool} \\
= : \forall\alpha, \beta \ \textbf{with} =: \alpha \rightarrow \alpha \rightarrow \mathit{bool} \, . \\
\qquad \mathit{pair}(\alpha, \beta) \rightarrow \mathit{pair}(\alpha, \beta) \rightarrow \mathit{bool} \\
= : \forall\alpha . \, \mathit{ref}(\alpha) \rightarrow \mathit{ref}(\alpha) \rightarrow \mathit{bool}
\end{array} \right\}
$$

The last assumption above specifies a polymorphic instance for =, reflecting that equality is meaningful for references (pointers).

Parametric assumption sets allow a limited form of recursion. If we define a dependency relation among identifiers in a type assumption set that says identifier f depends on g if and only if f has an assumption with a constraint on g, then we see that parametricity ensures that the transitive closure of the relation is antisymmetric and consequently mutual recursion is prohibited. For instance, the set in Figure 3 is mutually recursive and therefore is not parametric. Neither the assumptions for f nor g can be introduced because each requires the introduction of the other.

2.1 Regular Tree Languages

Problem *CS-SAT* has two inputs, A and C. In practice A usually varies little if at all across different instances of type inference. Thus we can benefit from suitably representing A and reusing its representation for different inputs C. A realistic measure of *CS-SAT* 's complexity should not ignore this fact. So although A is an assumption set, we assume that as an instance of *CS-SAT*, it is suitably represented. If A is parametric then every overloaded identifier x has an LCG of the form $\forall\alpha.\ \tau$ and the set of finite types π to which α can be instantiated, meaning one can derive $A \vdash x : \tau[\alpha := \pi]$, form a regular tree language.

Given an alphabet A, an A-valued tree t is specified by its set of nodes, or domain, $dom(t)$, and a valuation of the nodes in A. Formally, a *k-ary*, A-valued tree is a mapping $t : dom(t) \rightarrow A$ where $dom(t) \subseteq \{0, \ldots, k-1\}^*$ is a nonempty set and closed under prefixes. The frontier of t is the set of nodes $\{w \in dom(t) \mid \neg\exists i.wi \in dom(t)\}$. We assume that A is partitioned into a *ranked alphabet* Σ, and a *frontier alphabet* X. For any Σ and X, we denote the set of all finite ΣX-trees by $F_\Sigma(X)$.

Regular tree languages, or *forests*, can be characterized in different ways using tree recognizers (automata) [GeS84] or familiar operations of regular sets, like concatenation and closure, extended to finite sets of trees [Tho90]. To simplify our proofs, we choose to characterize them as forests generated by a class of context-free grammars called the regular tree grammars [GeS84].

Definition 2.3 *A regular ΣX-grammar G consists of*

- *a finite nonempty set N of nonterminal symbols,*
- *a finite set P of productions $A \rightarrow r$ where $A \in N$ and $r \in F_\Sigma(N \cup X)$, and*
- *an initial symbol $S \in N$.*

Definition 2.4 *If $G = (N, \Sigma, X, P, S)$ is a regular ΣX-grammar then the ΣX-forest generated by G is $T(G) = \{t \in F_\Sigma(X) \mid S \Rightarrow_G^* t\}$.*

From a given parametric assumption set A, the idea is to construct for each overloaded identifier x a regular tree grammar G_x such that if x has LCG $\forall\alpha.\ \tau$ then for any closed (variable-free) finite type π, $A \vdash x : \tau[\alpha := \pi]$ is derivable if and only if $\pi \in T(G_x)$. So determining whether constraint $x : \tau[\alpha := \pi]$ is satisfiable with respect to A amounts to parsing π. G_x always has a nonempty ranked alphabet of type constructors $\chi_1, \ldots, \chi_n$ and an empty frontier alphabet. So we drop the frontier alphabet from discussion and speak of just Σ-trees from now on, the collection of which is F_Σ for a given Σ.

Critical to our representation of a parametric overloading is the property that regular forests are effectively closed under intersection. This implies they are properly contained within the context-free languages since the latter are not closed under intersection.

Theorem 2.1 *If G_1 and G_2 are regular tree grammars then $T(G_1) \cap T(G_2)$ is generated by a regular tree grammar.*

Proof. Suppose $G_1 = (N_1, \Sigma, P_1, S_1)$ and $G_2 = (N_2, \Sigma, P_2, S_2)$ are regular Σ-grammars. Let Σ-grammar $G = (N_1 \times N_2, \Sigma, P, [S_1, S_2])$ where

$$[A, B] \rightarrow a([Y_1, Z_1], \ldots, [Y_n, Z_n]) \in P, \quad \text{for } n \geq 0$$

if and only if $A \rightarrow a(Y_1, \ldots, Y_n) \in P_1$, $B \rightarrow a(Z_1, \ldots, Z_n) \in P_2$, and $a \in \Sigma$. Then $T(G) = T(G_1) \cap T(G_2)$. □

Suppose x is overloaded in an initial parametric assumption set A with LCG $\forall\alpha.\ \tau$ and that Σ contains all type constructors of A. We construct G_x as follows. Since the overloading for x may be recursive, we first factor all assumptions on x into two sets, one containing its assumptions without any constraints on x and the other having its assumptions with only constraints on x if any. G_x then is the intersection of the regular Σ-grammars representing the two sets. These two tree grammars cannot depend on G_x since the transitive closure of the dependency relation is antisymmetric.

A regular Σ-grammar is constructed for each set as follows. For each assumption

$$x : \forall\gamma_1, \ldots, \gamma_n \textbf{ with } C\,.\,\tau[\alpha := \chi(\gamma_1, \ldots, \gamma_n)]$$

introduce n nonterminals $A_1, \ldots, A_n$ and create a production $S \rightarrow \chi(A_1, \ldots, A_n)$ such that A_i derives exactly $\bigcap_{k=1}^{m} T(G_{z_k})$ if γ_i appears in constraints on $z_1, \ldots, z_m$ in C and derives F_Σ otherwise. By Theorem 2.1, the intersection can be described by a regular Σ-grammar. Nonterminal S is the start symbol of the grammar. The finite types derivable from A_i correspond precisely to those types that satisfy all constraints in C involving γ_i.

For example, we construct regular Σ-grammars G_+ and G_* for the parametric assumption set in Figure 2. Let $\Sigma_0 = \{int, real\}$ and $\Sigma_1 = \{matrix\}$. Due to the constraint on $+$ needed to assert that $*$ may stand for matrix multiplication, construction of G_* depends on G_+. So we begin by factoring the assumptions for

$+$, leading to two regular tree grammars G_1 and G_2 where G_1 is

$$S_1 \rightarrow real \mid matrix(U)$$
$$U \rightarrow int \mid real \mid matrix(U)$$

and G_2 is

$$S_2 \rightarrow real \mid matrix(S_2)$$

G_1 arises from the assumptions for $+$ with the lone constraint on $+$ deleted. Therefore U derives F_Σ. G_2 on the other hand is constructed from the assumptions with only constraints on $+$ which in this example is the same as the original set. The regular Σ-grammar G_+ for $T(G_1) \cap T(G_2)$ becomes

$$[S_1, S_2] \rightarrow real \mid matrix([U, S_2])$$
$$[U, S_2] \rightarrow real \mid matrix([U, S_2])$$

Next we construct G_*. Corresponding to assumptions for $*$ without any constraints on $*$ is the grammar

$$S_3 \rightarrow int \mid real \mid matrix([S_1, S_2])$$

and to the assumptions with only constraints on $*$,

$$S_4 \rightarrow int \mid real \mid matrix(S_4)$$

G_* then represents their intersection and is given by

$$[S_3, S_4] \rightarrow int \mid real \mid matrix([[S_1, S_2], S_4])$$
$$[[S_1, S_2], S_4] \rightarrow real \mid matrix([[U, S_2], S_4])$$
$$[[U, S_2], S_4] \rightarrow real \mid matrix([[U, S_2], S_4])$$

Now if A denotes the set of Figure 2, then for any closed finite type τ, $A \vdash + : \tau \rightarrow \tau \rightarrow \tau$ is derivable if and only if $\tau \in T(G_+)$, likewise for $T(G_*)$. This actually follows from the next theorem which establishes the correctness of the representation.

Theorem 2.2 *If A is parametric and x is overloaded in A with LCG $\forall\alpha.\ \tau$ and regular Σ-grammar $G_x = (N, \Sigma, P, S)$ then $A \vdash x : \tau[\alpha := \pi]$ iff $\pi \in T(G_x)$.*

Proof. We use a normalized version of ML_o, replacing ($\forall$-elim) with rule ($\forall$-elim$'$):

$$\frac{x : \forall\bar{\alpha}\ \textbf{with}\ C\,.\,\tau' \in A, \quad A \vdash C[\bar{\alpha} := \bar{\tau}]}{A \vdash x : \tau'[\bar{\alpha} := \bar{\tau}]}$$

The normalized version and $ML_\circ$ are proved equivalent in [Smi91]. We prove $\pi \in T(G_x)$ implies $A \vdash x : \tau[\alpha := \pi]$ by induction on the structure of π:

($\pi = \chi$). If $\chi \in T(G_x)$ then $S \rightarrow \chi \in P$ which implies $x : \tau[\alpha := \chi] \in A$. By rule (hypoth) then $A \vdash x : \tau[\alpha := \chi]$.

($\pi = \chi(\bar{\tau})$). If $\chi(\bar{\tau}) \in T(G_x)$ then $S \rightarrow \chi(\bar{\gamma}) \in P$, $\tau_i \in T(G_i)$, where $G_i = (N, \Sigma, P, \gamma_i)$, and $x : \forall\bar{\gamma}\ \textbf{with}\ C\,.\,\tau[\alpha := \chi(\bar{\gamma})] \in A$. Suppose $z_1, \ldots, z_m$ are all identifiers constrained in C by γ_i. Since A is parametric, $z_k : \rho_k \in C$ implies z_k has $LCG\ \forall\gamma_i.\ \rho_k$ and z_k is overloaded in A. By the construction of G_x we have $T(G_i) = \bigcap_{k=1}^{m} T(G_{z_k})$ so $\tau_i \in T(G_{z_k})$ for $k = 1, \ldots, m$. By the inductive hypothesis, $A \vdash z_k : \rho_k[\gamma_i := \tau_i]$. So by rule ($\forall$-elim$'$), $A \vdash x : \tau[\alpha := \chi(\bar{\tau})]$.

Next we prove that $A \vdash x : \tau[\alpha := \pi]$ implies $\pi \in T(G_x)$ by induction on the length of the derivation of $A \vdash x : \tau[\alpha := \pi]$. The derivation ends with an application of rule (hypoth) or rule ($\forall$-elim$'$):

(hypoth). If $x : \tau[\alpha := \chi] \in A$ then $S \rightarrow \chi \in P$ which implies $\chi \in T(G_x)$.

($\forall$-elim$'$). The derivation ends with

$$\frac{x : \forall\bar{\gamma}\ \textbf{with}\ C\,.\,\tau[\alpha := \chi(\bar{\gamma})] \in A, \quad A \vdash C[\bar{\gamma} := \bar{\pi}]}{A \vdash x : \tau[\alpha := \chi(\bar{\pi})]}$$

Suppose $z_1, \ldots, z_m$ are all identifiers constrained in C by γ_i. Since A is parametric, $z_k : \rho_k \in C$ implies z_k has $LCG\ \forall\gamma_i.\ \rho_k$ and z_k is overloaded in A. Then $A \vdash C[\bar{\gamma} := \bar{\pi}]$ implies $A \vdash z_k : \rho_k[\gamma_i := \pi_i]$ so by the inductive hypothesis $\pi_i \in T(G_{z_k})$ for $k = 1, \ldots, m$, or $\pi_i \in \bigcap_{k=1}^{m} T(G_{z_k})$. Now $x : \forall\bar{\gamma}\ \textbf{with}\ C\,.\,\tau[\alpha := \chi(\bar{\gamma})] \in A$ implies $S \rightarrow \chi(\bar{\gamma}) \in P$. By virtue of the construction of G_x, we have $\pi_i \in T(G_i)$ and therefore $\chi(\bar{\pi}) \in T(G_x)$. □

3 CS-SAT is NP-hard for Parametric Overloading

The NP lower bound is proved by factoring a reduction from *3CNF-SAT* through the problem of computing the intersection of a sequence of regular forests. Though this is unnecessary and a simpler proof is possible, it is done in order to isolate the source of the hardness which lies in computing this intersection.

Theorem 3.1 *Given a parametric assumption set A with overloaded identifiers $x_1, \ldots, x_n$ whose assumptions are represented by regular tree grammars and a constraint set C over $x_1, \ldots, x_n$ such that $x : \rho \in C$ implies ρ is a generic instance of the LCG of x in A, deciding whether C is satisfiable under A is NP-hard.*

Proof. We give a P-time reduction from *3CNF-SAT*. Given a *3CNF* formula E, consisting of clauses

$d_1, \ldots, d_n$, we construct a parametric assumption set A_E, with all overloadings represented by regular tree grammars, and a constraint set C such that C is satisfiable under A_E if and only if E is satisfiable.

Suppose E has m distinct variables $x_1, \ldots, x_m$ and let the ranked terminal alphabet $\Sigma = \Sigma_0 \cup \Sigma_1$ where $\Sigma_0 = \{\epsilon\}$ and $\Sigma_1 = \{T, F\}$. Construct a regular Σ-grammar G_{d_i} for each clause d_i so that $\sigma \in T(G_{d_i})$ if and only if

$$\sigma = B_1(B_2(\cdots B_m(\epsilon)\cdots))$$

and the assignment of truth values $B_1, \ldots, B_m$ to $x_1, \ldots, x_m$ respectively satisfies d_i. If d_i contains variables x_j, x_k, and x_l, with $j < k < l$, and $x_j \leftarrow B_j$, $x_k \leftarrow B_k$, and $x_l \leftarrow B_l$ is a truth assignment satisfying d_i, then construct a regular Σ-grammar with start symbol x_1 and productions

$$x_j \rightarrow B_j(x_{j+1}) \quad x_k \rightarrow B_k(x_{k+1}) \quad x_l \rightarrow B_l(x_{l+1})$$

and for $1 \leq i \leq m$ with $i \neq j$, $i \neq k$, and $i \neq l$,

$$x_i \rightarrow T(x_{i+1}) \mid F(x_{i+1})$$

and finally $x_{m+1} \rightarrow \epsilon$. There is one such regular Σ-grammar for each of the 7 truth assignments satisfying d_i, call them $G_1, \ldots, G_7$. Then let

$$T(G_{d_i}) = \bigcup_{k=1}^{7} T(G_k)$$

G_{d_i} can be constructed in $O(m)$ steps so that for each nonterminal Y and truth value B, there is at most one production of the form $Y \rightarrow B(Z)$. For $1 \leq i \leq n$, add to A_E assumption $X_i : \epsilon$ if $X \rightarrow \epsilon$ is a production of G_{d_i} and assumption

$$Y_i : \forall\alpha \textbf{ with } Z_i : \alpha \,.\, B(\alpha)$$

if $Y \rightarrow B(Z)$ is a production of G_{d_i}. If G_{d_i} has start symbol S_i, then with

$$C = \{S_1 : \alpha, \ldots, S_n : \alpha\}$$

E is satisfiable if and only if $\bigcap_{i=1}^{n} T(G_{d_i})$ is nonempty, or if and only if C is satisfiable under A_E. □

As is the case for deciding whether a sequence of finite automata accept a common string, the source for the hardness of *CS-SAT* lies not in deciding emptiness but rather in computing the intersection, in this case, of a sequence of regular forests $T(G_1), \ldots, T(G_m)$. The emptiness of $T(G)$ for a regular tree grammar G is decidable in time $O(\mid G \mid^2)$ in the usual way. From the proof of Theorem 3.1 then every problem in NP is P-time Turing reducible to the problem of constructing the intersection of a sequence of regular tree grammars, so the construction is NP-hard. This helps to explain why the worst-case time complexity of an improved algorithm for computing the intersection of regular forests is still exponential [AiM91]. Actually computing the intersection is much harder. A weak PSPACE-hard lower bound follows immediately from the finite automaton intersection problem, treating strings as unary trees. A tighter exponential time lower bound follows from the complexity of the intersection problem for tree automata [FSV91]. For a fixed m, it can be computed in polynomial time.

4 Conclusion

Some might argue that given that ML typability is complete for DEXPTIME [KTU90], the fact that *CS-SAT* is NP-hard is insignificant. If we were concerned only about the worst-case time complexities of type inference algorithms then this might be true. But experience has shown that the DEXPTIME lower bound is not an issue in practice and type inference algorithms whose worst-case time complexities are exponential perform quite well on pratical programs. In fact it was folklore for many years that ML typability could be decided in polynomial time. So the complexity of *CS-SAT* could very well be the dominating complexity in practice. More experience is needed though in using systems like ML_o to determine whether the NP lower bound for *CS-SAT* is a practical limitation.

References

[AiM91] Aiken, A. and Murphy, B., "Implementing Regular Tree Expressions," *Proc. 5th Conf. on Functional Programming Languages and Computer Architecture*, LNCS 523, Springer-Verlag, pp. 427–447, 1991.

[CHO92] Chen, K., Hudak, P. and Odersky, M., "Parametric Type Classes," *Proc. 7th ACM Conf. on Lisp and Functional Programming*, pp. 170–181, 1992.

[CDO91] Cormack, G. Duggan, D. and Ophel, J., "Decidable Type Reconstruction with Recursive Overloading," Extended Abstract, Department of Computer Science, University of Waterloo, 1991.

[CoD78] Coppo, M. and Dezani-Ciancaglini M., "A New Type Assignment for λ Terms," *Archi. Math. Logik*, Vol. 19, pp. 139–156, 1978.

[CDV80] Coppo, M., Dezani-Ciancaglini M. and Venneri, B., "Principal Type Schemes and Lambda Calculus Semantics," In *To H.B. Curry: Essays on Combinatory Logic, Lambda Calculus, and Formalism*, Academic Press, pp. 535–560, 1980.

[DaM82] Damas, L. and Milner, R., "Principal Type Schemes for Functional Programs," *Proc. 9th ACM Symposium on Principles of Programming Languages*, pp. 207–212, 1982.

[FSV91] Frühwirth, T., Shapiro, E., Vardi, M. and Yardeni,E., "Logic Programs as Types for Logic Programs," *Proc. 6th Annual Symp. on Logic in Computer Science*, pp. 300–309, 1991.

[GeS84] Gecseg, F. and Steinby M., Tree Automata, Akademiai Kiado, Budapest Hungary, 1984.

[Jon92] Jones, M., "A theory of qualified types," *Proc. 4th European Symposium on Programming*, LNCS 582, Springer-Verlag, pp. 287–306, 1992.

[Kae88] Kaes, S., "Parametric Overloading in Polymorphic Programming Languages," *Proc. 2nd European Symposium on Programming*, LNCS 300, Springer-Verlag, pp. 131–144, 1988.

[Kae92] Kaes, S., "Type Inference in the Presence of Overloading, Subtyping, and Recursive Types," *Proc. 7th ACM Conf. on Lisp and Functional Programming*, pp. 193–204, 1992.

[KTU90] Kfoury, A., Tiuryn, J. and Urzyczyn, P., "An Analysis of ML Typability," *Proc. 15th Colloquium on Trees in Algebra and Programming*, LNCS 431, Springer-Verlag, pp. 206–220, 1990.

[Lie90] Lievant, D., "Discrete Polymorphism," *Proc. 1990 ACM Conf. on Lisp and Functional Programming*, pp. 288–297, 1990.

[McC84] McCracken, N., "The Typechecking of Programs with Implicit Type Structure," *Semantics of Data Types* LNCS 173, pp. 301–315, 1984.

[Mil78] Milner, R., "A Theory of Type Polymorphism in Programming," *J. of Computer and System Sciences*, Vol. 17, pp. 348–375, 1978.

[Pie91] Pierce, B., "Programming with Intersection Types and Bounded Polymorphism," Ph.D. Thesis, Department of Computer Science, Carnegie Mellon University, Technical Report CMU-CS-91-205, 1991.

[Rey70] Reynolds, J.C., "Transformational Systems and the Algebraic Structure of Atomic Formulas," *Machine Intelligence*, Vol. 5, pp. 135–151, 1970.

[Rey88] Reynolds, J.C., "Preliminary Design of the Programming Language Forsythe," Report CMU-CS-88-159, Carnegie Mellon University, 1988

[Sal78] Sallé P., "Une Extension de la Théorie des Types en λ-calculi," LNCS 62, Springer-Verlag pp. 398–410, 1978.

[Smi91] Smith, G.S., "Polymorphic Type Inference for Languages with Overloading and Subtyping," Ph.D. Thesis, Department of Computer Science, Cornell University, Technical Report 91-1230, 1991.

[Smi93] Smith, G.S., "Polymorphic Type Inference with Overloading and Subtyping," *Proc. TAPSOFT '93*, LNCS 668, Springer-Verlag, pp. 671–685, 1993.

[Tho90] Thomas,W., "Automata on Infinite Trees," Handbook of Theoretical Computer Science, Volume B, Formal Methods and Semantics, J. van-Leeuwen, Ed. pp. 165–184, 1990.

[WaB89] Wadler, P. and Blott, S., "How to make *ad-hoc* polymorphism less *ad-hoc*," *Proc. 16th ACM Symposium on Principles of Programming Languages*, pp. 60–76, 1989.